

dynRDF: Using Deep Contextual Bandits to Optimize Position Flooding in Urban UAV Networks

Konrad Fuger 

*Institute of Communication Networks
Hamburg University of Technology
Hamburg, Germany
Email: k.fuger@tuhh.de*

Kwame Ofori

*Netlight Consulting GmbH
Hamburg, Germany*

Email: kwame.ofori@netlight.com

Andreas Timm-Giel 

*Institute of Communication Networks
Hamburg University of Technology
Hamburg, Germany
Email: timm-giel@tuhh.de*

Abstract—Advances in mechanical capabilities and mass manufacturing of Unmanned Aerial Vehicles (UAVs) are driving their application in various fields from precision agriculture to infrastructure monitoring and on-demand parcel delivery. Especially in urban areas it is projected that large amount of UAVs will inhabit the airspace. To facilitate the safe and reliable operation of large-scale urban UAV deployments, an Unmanned Aerial Traffic Management (UTM) system is required. Such a system needs to be aware of all movements within the airspace to control and monitor urban UAV operations. One way to realize this is the establishment of an ad-hoc network, which UAVs use for network-wide dissemination of their positions. Recently, Rate Decay Flooding (RDF) has been proposed as a tailor-made protocol to realize such a system. Although RDF has been proven to be efficient in supporting UTM applications in larger networks than ordinarily possible, much of its success relies on the proper selection of protocol parameters. In this work, we propose a reinforcement-learning framework that automatically adapts the configuration of RDF to its perceived environment. We utilize deep contextual bandits as a light-weight, but effective method to capture the non-linear relationship between the perceived environment and the achieved performance. We name this extension Dynamic Rate Decay Flooding (dynRDF). In a simulation study, we show that this solution is effective in finding optimal configurations for RDF for varying network sizes. To achieve this, only 2.7 % of all possible configurations had to be explored. Allowing dynRDF to also take the local UAV density into account, a performance gain of more than 12 % is achieved in a relevant composite metric capturing both the timely dissemination of position updates to nearby UAVs and reliable network-wide dissemination.

Index Terms—UAV, UTM, Flooding, Reinforcement Learning, Deep Contextual Bandits

I. INTRODUCTION

The use of Unmanned Aerial Vehicles (UAVs) has been proposed for many industries ranging from precision agriculture over various sensing application up to the inspection and maintenance of critical infrastructure. Among these, the application with the highest economic potential is the on-demand delivery of goods in urban areas. The realization of

This work has been partially funded by the German Federal Ministry of Economic Affairs and Climate Action (BMWK, LuFo VI, 20Q1939C).

this application would alleviate our congested urban roads from all or most delivery traffic as parcels are autonomously delivered by UAVs swarming over our rooftops. But as studies show, thousands of UAVs are required in large urban centers to make this a reality [1], [2]. For a safe and reliable operation, these UAVs need to be monitored and their movements managed. Additionally, they need to be aware of their surroundings to self-separate and avoid in-air collisions. A system offering these services is called an Unmanned Aerial Traffic Management (UTM). While there are multiple concepts for the exact operation of an UTM, the availability of position information, identities and intents of all UAVs in the airspace is a necessity for any of them.

Over the years many protocols have been proposed to facilitate network-wide information exchange in high dynamic ad-hoc networks. Recently, Rate Decay Flooding (RDF) was proposed as a solution specifically tailored to large-scale urban to UAV networks, supporting networks twice the size compared to traditional approaches [3]. Nevertheless, as with most communication protocol, the correct parameterization is crucial to achieve this performance. It was shown that the selection of the optimal parameters strongly depends on external factors such as the size of the supported network and its density. To overcome this issue, we propose a protocol extension to RDF, which dynamically adjusts its parameters based on local measurements and a learned policy of optimal parameterization. The main contributions of this paper can be summarized as follows:

- We propose Dynamic Rate Decay Flooding (dynRDF) as an extension to RDF to dynamically change protocol configurations for changing network characteristics.
- We developed a reinforcement learning framework using deep contextual bandits to efficiently obtain an optimal policy for dynRDF.
- The efficacy of the proposed system is evaluated with an extensive simulation study.

Our results provide a crucial step towards the implementation of RDF in the real world where urban UAV networks rarely come with in fixed network sizes and static node

densities.

The remainder of this paper is structured as follows: In Section II we present related work on the topic of protocol parameter optimization in general and the application of reinforcement learning in the field of UAV communication. With this background, we outline the working principle of dynRDF in Section III. Our reinforcement learning setup is explained in Section IV. We present a performance evaluation of the proposed system in Section V after which we conclude this paper in Section VI.

II. RELATED WORK

Many modern communication protocols rely on a set of finely tuned parameters that vastly improve the performance of any given protocols. Often reasonable values for these parameters can be identified with extensive simulation studies and refined with real-world experiments. While this is a trusted method, the effort required increases sharply with the parameter space to be explored. Alternatively, if a mathematical model of a given protocol is available, the optimal configuration can quickly be found in vast (or even infinite) parameter spaces. Nevertheless, this requires highly accurate models incorporating all important aspects of reality. In lieu of accurate models, iterative optimization techniques such as genetic algorithms, particle swarm optimization or simulated annealing [4] and game-theory based approaches [5] have been applied.

Recently, machine learning has become the favorite method for parameter optimization as machine learning models can describe the effect of a given parameter on the performance of a protocol without the need for a handcrafted mathematical model and more efficiently than brute-force simulations [6]. In particular Reinforcement Learning (RL) has gained a lot of prominence due to its capability to effectively explore very large parameter spaces by balancing exploration of unknown choices with exploitation of the known best performing parameters [7]. Opposed to brute-force methods, RL prioritizes parameter choices that promise to yield better results. Therefore, RL methods are significantly faster in converging towards the optimal set of parameters. Also applied to the field of UAV communication, RL has gained some prominence: For example in [8] a Q-Learning approach is used to optimize the delay and energy consumption in an UAV assisted wireless network. In [9] a deep Q-Learning approach was used to optimize the positioning of UAVs within a swarm to increase their throughput. While these approaches would be applicable for the scenario at hand, Q-Learning supposes that an action taken by the algorithm can influence the state of the system, which is not the case in our problem formulation as we will see. For problems like ours (contextual) multi-armed bandits are an effective approach. In the field of UAV communication they have for example been used in [10] to optimize the parameters of LoRa-based communication between UAVs and a gateway applying different variants of multi-armed bandits. To optimize the parameters of a rate adaption mechanism in

a WiFi connected UAV network a contextual bandit is used in [11], which takes the channel state into account to set its parameters. Nevertheless, if the underlying relationships to be learned are more complex classic contextual multi-armed bandits become inefficient and deep contextual bandits become the method of choice. While these have been successfully applied in many domains, e.g. [12], they remain a rare solution for the optimization of communication protocols.

III. DYNAMIC RATE DECAY FLOODING

RDF is a tailor made protocol for position flooding in large-scale UAV networks to realize the most challenging UTM applications. It is build upon the insight, that position information becomes less important the further it travels from its source. While conventional flooding protocols scale poorly and are often at risk to overload the network [13], RDF progressively delays packet forwarding on every hop and thereby shields the network from overload. By using RDF every UAV periodically emits a position report with a fixed send interval i_0 . All UAVs receiving the packet calculate a timer so that a forwarding interval of $i_0 \cdot h^q$ is maintained for every particular source, where h is the hop count of the packet and q is a pre-configured parameter called decay factor. After this timer expires, all UAVs set a second timer inversely proportional to the distance of the sender as proposed in [14], by using the position data included in the packet and their own position obtained through GNSS. Whenever an UAV overhears the forwarding of a packet, it cancels its corresponding timer and abstains from forwarding as another node in a more favorable position has already done so. This reduces redundant transmissions. In our extension dynRDF, we allow UAVs to select i_0 and q based on a policy (described below). For this, whenever a packet is sent by an UAV using dynRDF, it consults its policy about the current send interval i_0 and schedules its next packet transmission i_0 in the future. Additionally, the current decay factor q is obtained form the policy. The decay factor has to be applied by all forwarding UAVs while the packet travels through the network. Therefore, the sender adds its decay factor together with the current send interval into a header field of the emitted packet. Configuring dynRDF with a constant policy, that is a policy that always returns the same i_0 and q , yields the behavior of RDF. In the original study of RDF a default configuration of $i_0 = 120$ ms and $q = 2.0$ was obtained through brute-force simulations over a coarse configuration space. We refer to this default configuration as const. (120 | 2.0) and use it as a benchmark. With this configuration a network roughly twice the size compared to conventional approaches could be supported. Nevertheless, the study also shows that for smaller networks other configurations yield better results.

To measure the performance of dynRDF, we use two metrics to describe the performance of a position flooding protocol for UTM as were introduced in [3]: The *Excess Probability* P_{EX} quantifies the ability of a flooding protocol to deliver timely position updates to near-by UAVs. For

this, a time threshold $T_{\text{CADA}} = 736$ ms was established for Communication Aided Detect and Avoid (CADA) adapted from a simulation study for UAV avoidance [15]. If a position update ages beyond that this threshold without the reception of a more recent position update, an abort maneuver has to be executed. The excess probability is defined as the number of position updates exceeding this threshold divided by the total number of position updates received. A threshold of $\Gamma_{\text{EX}} = 1\%$ was proposed for successful CADA.

Complementary to this, the *Dissemination Rate* P_D describes how reliable a flooding protocol can distribute position updates network wide. As this is relevant for airspace monitoring (AM), another threshold T_{AM} was established¹. Successful dissemination between two UAVs is achieved if at least one position update is received from the other UAV within this threshold. The dissemination rate is calculated as the quotient of the number successful disseminations divided by total number of possible disseminations. For successful airspace monitoring a threshold of $\Gamma_D = 99\%$ must be exceeded.

A. Deep Contextual Bandit

To obtain a policy to be used in dynRDF we employ a deep contextual bandit. Deep contextual bandits are a simple yet powerful form of RL [16]. As usual in RL settings, the problem is formulated as an environment that can be in any state $S_t \in \mathcal{S}$. An agent operates on this environment by executing an action $A_t \in \mathcal{A}$ on the environment which yields a reward r_{t+1} and presents itself in a new state S_{t+1} . The relationship between states, actions and rewards are captured by a value function $Q(S, A)$. Different from other RL approaches, the value function only captures the expected reward obtained from the environment given the current action and state and does not try to incorporate the next states value. At every stage of the learning process, a greedy policy π_g is obtained from the learned value function mapping states to the action promising the highest reward:

$$\pi_g(S) = \underset{A \in \mathcal{A}}{\operatorname{argmax}} Q(S, A) \quad (1)$$

This greedy policy maximizes the expected reward by following the predicted best action for any given state. However, to sufficiently learn the *true* value function, also actions deemed sub-optimal by the current value function need to be explored. To do so, we generate exploratory policies π_e by superimposing the current value function with Gaussian distributed noise with a given standard deviation σ .

$$\pi_e(S) = \underset{A \in \mathcal{A}}{\operatorname{argmax}} [Q(S, A) + \mathcal{N}(0, \sigma)] \quad (2)$$

With this approach using a small value of σ allows for the exploration of policies that are close to the current greedy policy while large values of σ promote the exploration of widely different policies. Usually exploration of novel

¹ T_{AM} varies with the network size and is defined, so that dissemination must be achieved before any UAV can cross the monitored network.

policies and exploitation of the current greedy policy have to be balanced to maximize the rewards obtained during the learning phase. For the application of dynRDF we only aim to derive an optimal policy once the learning has converged, allowing us to widely explore policies during the learning phase, even if they are unlikely to yield a high reward.

The state in our scenario that is presented to the deep contextual bandit consists of:

- N : total number of nodes in the network,
- N_{1H} : number of one-hop neighbors,
- N_{2H} : number of two-hop neighbors.

Actions consist of the parameter pair (i_0, q) to be applied when sending out position updates. Every UAV constantly monitors its current state and applies actions according to the policy π . Crucially, the state of a given UAV during the simulation does not depend on the actions taken. This makes deep contextual bandits a natural fit for our problem formulation as this method does not consider the progression of states over time. The two Key Performance Indicators (KPIs) P_D and P_{EX} are measured over a longer time-horizon involving many visited states and applied actions, before values are obtained and a single reward is calculated. This is a special case of deep contextual bandits where usually the execution of an action yields an immediate reward. For position flooding in UAV networks the performance cannot be meaningfully measured from a single action and therefore only a summary reward can be obtained for evaluating a complete policy.

The learning objective is to discover a policy that ensures $P_D \geq \Gamma_D$ and $P_{\text{EX}} \leq \Gamma_{\text{EX}}$ at the same time. This is encoded in the reward function $R(P_{\text{EX}}, P_D)$ that transforms our KPIs into a scalar reward:

$$R(P_{\text{EX}}, P_D) = \begin{cases} \frac{1}{2} \cdot \left(\frac{P_D}{\Gamma_D}\right)^2 \cdot \left(\frac{1 - P_{\text{EX}}}{1 - \Gamma_{\text{EX}}}\right)^2, & \text{A} \\ \frac{1}{2} \cdot \left(\frac{1 - P_{\text{EX}}}{1 - \Gamma_{\text{EX}}}\right)^2, & \text{B}_1 \\ \frac{1}{2} \cdot \left(\frac{P_D}{\Gamma_D}\right)^2, & \text{B}_2 \\ \frac{1}{2} + \frac{1}{2} \cdot \frac{P_D - \Gamma_D}{1 - \Gamma_D} \cdot \frac{\Gamma_{\text{EX}} - P_{\text{EX}}}{\Gamma_{\text{EX}}}, & \text{C} \end{cases} \quad (3)$$

where the conditions are defined as

$$\begin{aligned} \text{A} & : P_D < \Gamma_D \wedge P_{\text{EX}} > \Gamma_{\text{EX}} \\ \text{B}_1 & : P_D \geq \Gamma_D \wedge P_{\text{EX}} > \Gamma_{\text{EX}} \\ \text{B}_2 & : P_D < \Gamma_D \wedge P_{\text{EX}} \leq \Gamma_{\text{EX}} \\ \text{C} & : \text{otherwise} \end{aligned} \quad (4)$$

This function is schematically depicted in Figure 1 and can be subdivided into four regions: In region A, neither of the KPI targets is met and a reward between 0 and 0.5 is awarded where better performance yields better rewards. In regions B₁ and B₂ one KPI target is met but not the other one. Here

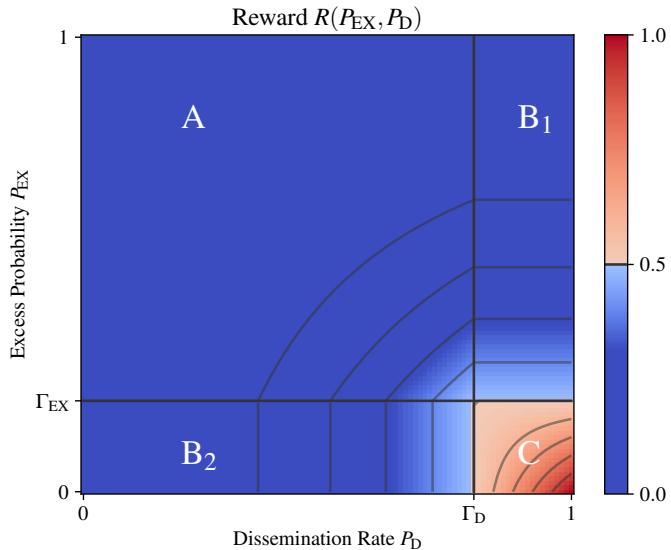


Figure 1: Reward function mapping P_{EX} and P_D to a scalar reward r (not to scale).

we again yield a reward of up to 0.5. The reward function is designed so that a higher reward can only be achieved if the KPI that does not meet its target yet, is improved. Finally, in region C both targets are met and a *minimum* reward of 0.5 is obtained. With this the threshold of $r = 0.5$ separates rewards obtained from successful policy from these obtained from unsuccessful ones.

B. Deep Neural Network

In deep contextual bandits, the value function $Q(S, A)$ is approximated by a deep neural network. The structure it is displayed in Figure 2. It is a fully connected neural network where the state-action tuple $(S, A) = (\{N, N_{1H}, N_{2H}\}, \{i_0, q\})$ is fed into the input layer. Five hidden layers, each consisting of 20 units are used. Finally, the Q-value Q is predicted by a single unit in the output

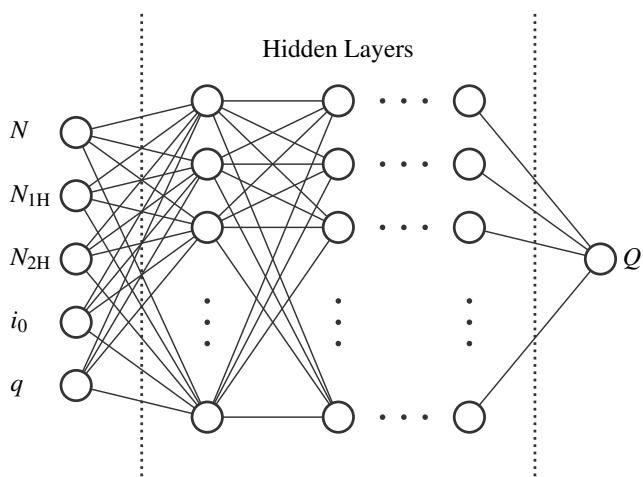


Figure 2: Architecture of the Deep Neural Network.

layer. Each layer is followed by a ReLu activation function to introduce non-linearity. To learn the value function, we generate experiences of state, action and reward (S_t, A_t, r_t) in simulation. The neural net is trained to represent the value of a given state-action pair (S, A) as the expected reward $Q(S, A) = \mathbb{E}[r_t]$ obtained over all experiences. In contrast to regular contextual bandits, using a neural network to represent the value function allows deep contextual bandits to generalize the value function also for unseen state-action pairs.

Our model contains 1821 trainable parameters. While it would be feasible to employ much larger models, our results show that our model is effective to represent a sufficient estimate of the value function.

IV. REINFORCEMENT LEARNING SETUP

As the KPIs for position flooding in UAV networks are summary metrics that are computed over long periods of time and averaged over many UAVs, the problem at hand presents itself as a very sparse learning environment. In particular, each simulation needs to be run to completion to produce values for P_D and P_{EX} and in turn yield a single reward r . Especially for larger networks, these simulations can easily last for more than 5 h. To make learning feasible under these constraints, an efficient setup is required that leverages highly parallel simulations to generate sufficient samples for reinforcement learning.

To achieve that, we apply round based updates of the neural network in a deep contextual bandit learning loop inspired by generalized policy iteration. The round based approach consists of two alternating phases: During policy evaluation we use a highly parallelized simulation setup to evaluate the performance of a given policy. In the policy improvement phase, we use the obtained results to re-train our neural network and obtain an improved greedy policy π_g with respective exploration policies π_e . To start the process, an initial, random policy is generated and marked for evaluation to initiate the first policy evaluation.

A. Policy Evaluation

When the policy evaluation phase starts, all policies marked for evaluation are fetched and simulations are prepared using these policies. dynRDF is implemented in a ns-3-based [17] simulator. IEEE 802.11p [18] is used as a link-layer with the Friis channel model [19] modelling favorable channel characteristics for UAVs over the roofs of cities. In absence of available real-world traces, random direction mobility is used. Based on the number of UAVs, the network size is configured so that a density of 12 UAVs/km² is maintained representing an urban parcel-delivery scenario. Each simulation is repeated 20 times with different random seeds to account for the stochastic behavior of the simulation results. Whenever a simulation run terminates, its results are stored in a database for subsequent policy improvement. These results include a unique identifier for the evaluated policy, all states observed during the simulation including

their relative frequency, the applied actions and finally the obtained values for P_{EX} and P_{D} .

B. Policy Improvement

As soon as all simulation runs are completed, the policy improvement phase begins by calculating the respective rewards $r = R(P_{\text{EX}}, P_{\text{D}})$ for each simulation. The resulting state-action-reward dataset is then used to retrain the neural network. Training is performed on the full dataset in a single batch. The neural network parameters are updated using the ADAM optimizer [20] with a mean squared error loss function, which minimizes the difference between the predicted Q-value and the observed Q-value, i.e., the average reward r for each state-action pair.

The updated greedy policy π_g is generated by selecting the action that yields the maximum predicted Q-value for each state. To balance exploration and exploitation, we additionally generate nine exploration policies $\pi_e^{(i)}, i = 1..9$. These exploration policies are created using the noise-based process outlined in Section III-A. The exploration policies $\pi_e^{(i)}$ are generated with a decaying noise floor $\sigma(i)$, designed to promote exploration of alternative actions and escape local optima:

$$\sigma(i) = \frac{\sigma_\theta}{i}, \quad \sigma_\theta = 3.0 \quad (5)$$

The varying noise floor ensures that exploration remains focused around the greedy policy π_g , while still allowing for deviations that potentially uncover other effective actions. At the end of the policy improvement step, ten policies are available: the greedy policy π_g and the nine exploration policies $\pi_e^{(1)}, \dots, \pi_e^{(9)}$. All policies are stored in a database and marked for evaluation, which in turn triggers the policy evaluation phase.

V. RESULTS

To evaluate the proposed solution, we devised two experiments exploring the effectiveness of dynRDF to learn both a policy for varying network sizes or UAV density. In both scenarios, we consider the same action space $\mathcal{A} = \{i_0, q\}$ where

$$i_0 [\text{ms}] \in [30, 180], \quad i_0 \in \mathbb{N}_0 \quad (6)$$

and

$$q \in k \cdot 0.05, \quad k \in \mathbb{N}_0, \quad q \leq 3.0 \quad (7)$$

resulting in an action space of size $|\mathcal{A}| = 9211$. Both experiments were conducted according to the setup described above.

A. Scenario I: Changing Network Sizes

In our first scenario, we limit the state space to only N , thereby learning the dependency between network size and optimal RDF configuration. All policies are evaluated with an equal amount of simulation runs for networks between $N = 100$ and 500 nodes. The evolution of the learning

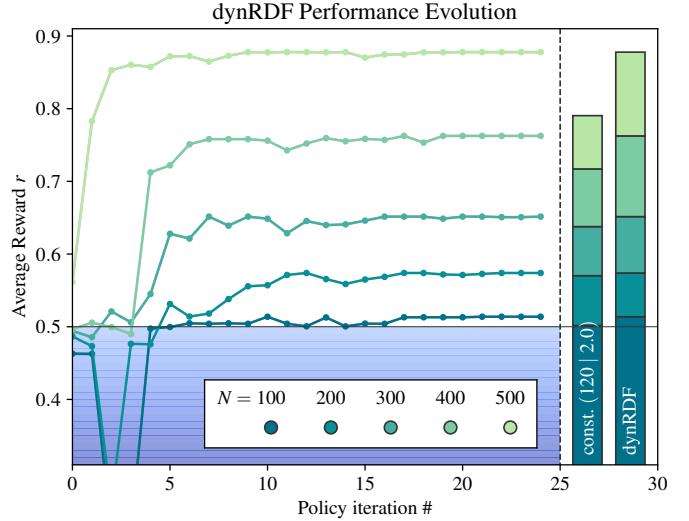


Figure 3: Evolution of the achieved reward for dynRDF for network sizes between 100 and 500.

process is depicted in Figure 3. Here, the solid lines signify the average achieved reward for networks of a given size under the greedy policy π_g , while the bars to the right represent the performance gain against the baseline configuration $\text{const. } (120 | 2.0)$, that was established in previous work by a coarse brute-force simulation study as the configuration that allows for the largest possible network size to be supported. Additionally, we have shaded the region of rewards less than 0.5 separating acceptable from unacceptable performance. Initially, our deep contextual bandit is randomly initialized, and therefore the performance is insufficient for all but the smallest network size ($N = 100$). In the first policy iterations afterwards the performance decreases further as the system did not gather enough experience to predict the performance accurately. Only after the fourth iteration the performance steeply rises for all network sizes. While the performance remains on a relatively high and stable level, it takes up to iteration 20 to achieve a performance gain for all network sizes. Afterwards the system converged as no change of performance is experienced. For a network of $N = 100$ nodes, a performance gain of 11% in reward is achieved compared to the baseline configuration. As the baseline was optimized for larger networks, it is no surprise that the gain decreases for larger networks. Still, averaged over all network sizes dynRDF achieves a 4.5% higher reward.

While an improvement in reward indicates better performance it is important to compare dynRDFs optimal policy to the RDF baseline in terms of the dissemination rate P_{D} and excess probability P_{EX} . This is shown in Figure 4 including 95% confidence intervals. Here we can see, that for networks between 100 and 400 nodes, RDF and dynRDF achieve almost the same dissemination rate with RDF being a fraction of a percent higher. At the same time, dynRDF manages to strongly decrease the excess probability (lower is better). For networks of 100 nodes, the excess probability

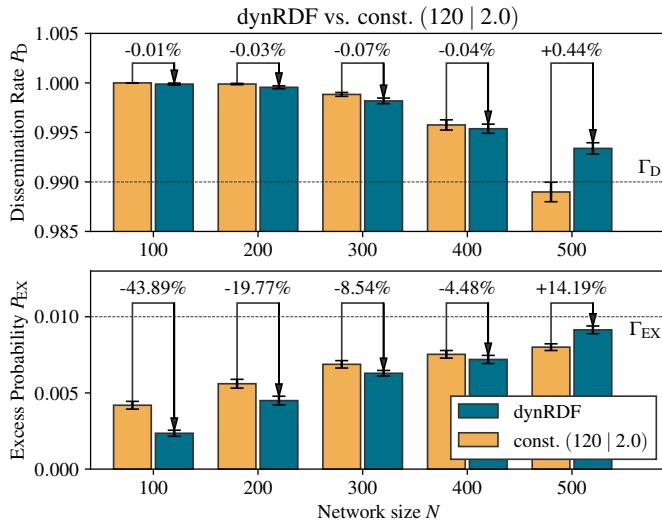


Figure 4: Comparison of dynRDFs optimal policy to RDFs baseline policy.

is almost halved. While this advantage diminishes for larger networks, a solid decrease in excess probability is maintained for networks up to 400 UAVs. The picture only changes for our largest network of 500 UAVs. Here, RDF barely misses the performance target Γ_D even considering the confidence interval and therefore does not support this network size². This is even though we have seen an average reward just above 0.5 for networks of this size, but since our reward function is non-linear, the average reward is not equal to the reward computed with the average KPIs for a given scenario. In contrast, the optimal policy of dynRDF manages to achieve a slightly higher dissemination rate so that the performance target is met. This improvement comes at the expense of an increase in excess probability by 14 %. But as the excess probability remains below our performance target Γ_{EX} , this trade off is fruitful to improve the system performance and support larger networks. These results demonstrate the ability of dynRDF to adapt to different situations: Driven by our reward function, dynRDF minimizes the excess probability to achieve a higher reward as long as the performance targets for both KPIs are met. Only once one of the KPIs is at risk of falling below the threshold, dynRDF dynamically prioritizes this KPI as our reward function foremost encourages that both KPI targets are met.

While this proves that dynRDF was indeed able to converge on an optimal policy that achieves a performance gain for all network sizes simultaneously, more insight can be obtained by looking at the actual policy derived. This is presented in Figure 5 where we show the action space. For each network size, we have marked the optimal action as prescribed by the final greedy policy π_g . It can be seen that for networks of $N = 100$ UAVs, RDF is optimally configured

²In the original simulation study of RDF networks up to 525 UAVs could be supported [3]. In our simulation we use slightly larger headers to carry the decay factor q and send interval i_0 which explain the discrepancy.

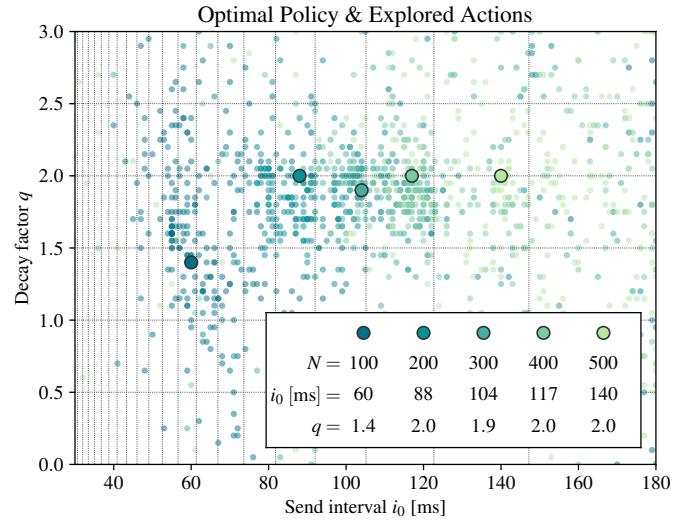


Figure 5: Optimal policy found by dynRDF and scatter plot of explored actions.

with a send interval of 60 ms and a decay factor of 1.4. For larger networks, the send interval gradually increases while an optimal decay factor around $q = 2.0$ is found. Generally speaking, RDF must always be configured so that the generated traffic does not overload the network. In any given network this can be achieved by either increasing the send interval or increasing the decay factor. From these results it becomes apparent that (apart from very small networks) it is always beneficial to reduce the traffic by increasing the send interval instead of increasing the decay factor beyond 2.0. To highlight the efficiency of our approach, we have used the same figure to mark all explored actions during the learning process. While we see that for all network sizes explored actions can be found all over the action space, they mostly concentrate around the optimal actions as intended with our noise-based exploration. Over all considered network sizes, only 2.7 % of the action space needed to be explored before an optimal policy was found.

B. Scenario II: Adapting to Local UAV Density

In our second experiment, we have fixed the network size to $N = 200$, but enabled the state parameters N_{1H} and N_{2H} representing the one- and two-hop neighbors of a given node respectively. The intuition here is that not all UAVs using dynRDF in a network of a given size need to use the same configuration. For example, UAVs that have fewer neighbors are likely to be positioned in border regions of the network where a lower packet loss is experienced, but the highest number of hops is required for a packet to traverse the network.

To explore whether this diversity can be exploited, we ran simulations of RDF for networks of $N = 200$ UAVs with the optimal configuration found in the previous experiment. The joint distribution of states is shown in Figure 6 with marginal distributions on the sides. Here we can clearly observe a that

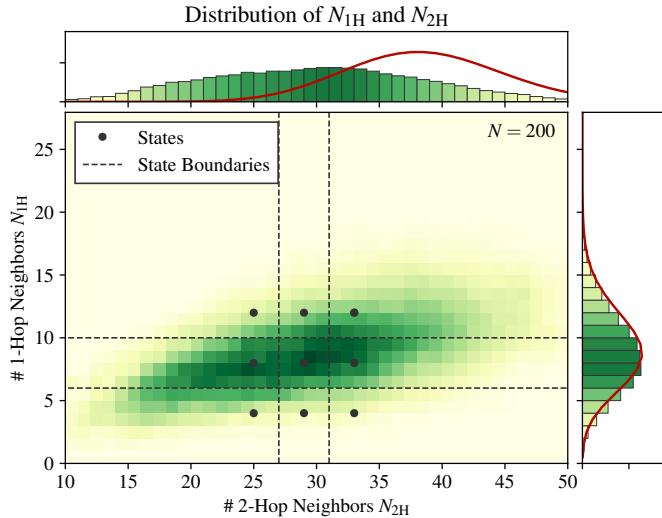


Figure 6: Distribution of N_{1H} and N_{2H} states as experienced in a network of 200 UAVs.

individual UAV experience a wide variety of states. For the one-hop neighbors N_{1H} the marginal distribution precisely follows a Poisson distribution (red line) as is expected since we simulate a scenario with a uniform UAV distribution in space. Interestingly, this cannot be observed for the two-hop neighbors N_{2H} . This is an effect of the limited simulation area: UAVs in the center of a simulation setting measure values of N_{2H} following a Poisson-Distribution, UAVs closer to the edge see less two-hop neighbors as some of their two-hop neighborhood is empty. While this could be considered a simulation artifact, real world large-scale UAVs networks will also exhibit this behavior as every network will have a border region with less neighboring UAVs that could be exploited by dynRDF. Unsurprisingly, we also see a correlation between N_{1H} and N_{2H} as UAVs in dense regions of the network tend to have both many one- and two-hop neighbors. We partition the state space into nine regions as indicated by the black dots and dashed lines in the figure so that dynRDF is able to learn an independent action for each of the regions.

Again, we execute our learning framework, now simulating only networks with $N = 200$ UAVs. As earlier, before every simulation phase, the current greedy policy π_g as well as nine exploratory policies π_e are synthesized and simulations are executed for every policy. Every UAV in the simulation now constantly monitors its state in terms of N_{1H} and N_{2H} and consults the synthesized policy for the corresponding action to take.

The progression of the learning in this experiment is displayed in Figure 7. Here we show the performance of the current greedy policy π_g over time (solid line) as well as the predicted performance (dashed line) and the performance of the exploration policies π_e as colored dots based on their value of σ . Additionally, we show the performance of the optimal action for networks of 200 UAVs obtained in the previous experiment const. (88 | 2.0) as a benchmark.

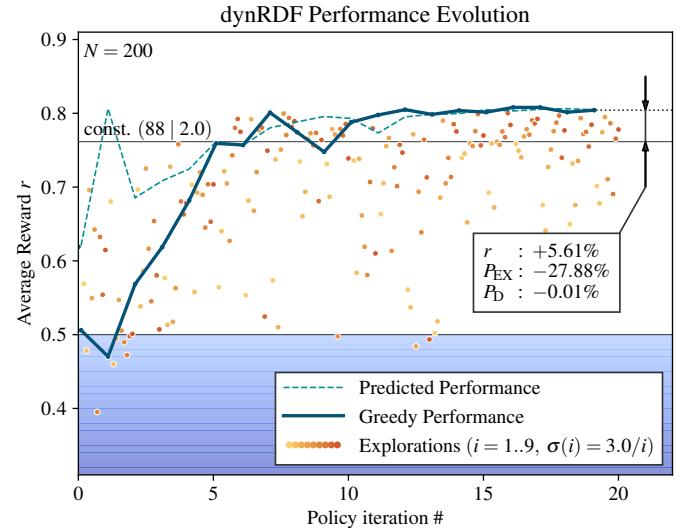


Figure 7: Evolution of the achieved reward for dynRDF for a network size of $N = 200$.

Initially, dynRDF is randomly initialized and performs poorly compared to the benchmark configuration. We also see that the predicted reward and the achieved reward show a strong discrepancy. It takes five iterations until the learned greedy policy performs comparable to the benchmark. Here, for the first time, we see exploratory policies consistently outperforming the benchmark. Nevertheless, it takes roughly ten more iterations for dynRDF to converge to an optimal policy as indicated by the fact that predicted reward closely matches achieved reward while no better-performing exploratory policies are found anymore. This experiment also highlights the effectiveness of the employed exploration strategy: In the beginning, when the value function is randomly initialized, a wide range of actions is explored shown by the wide range of achieved performance. Only later we see explorations with a small value of σ (indicated by darker dots) cluster closely around the greedy performance while larger values of σ still result in explorations vastly different from the current optimum. For a network of $N = 200$ UAVs the current benchmark already outperforms RDFs default configuration by 6.3 %. Allowing dynRDF to optimize for the local state of each individual UAV increases the reward by a further 5.6 % yielding a combined gain of 12.1 % compared to const. (120 | 2.0).

The state-action mapping of the derived policy is listed in Table I. Comparing the actions to the optimal action found in the first experiment, we can conclude that dynRDF has found actions close to the previously found optimal action both in send interval and decay factor. Nevertheless, we see some diversity in the policy: The used send intervals range from 67 ms to 103 ms and the decay factor varies between 1.70 and 2.70. While dynRDF likely exploits complex interactions between UAVs in different states, we can still make some interesting observations about the obtained policy: For example, whenever we are in the $N_{1H} = 12$

state, a decay factor larger than 2.0 is selected, highlighting the need to reduce the data traffic in scenarios with a high node density. Additionally, it is notable, that only in the state with the highest node density a send interval higher than the benchmark was selected. Finally, we have measured how often each state was observed and the corresponding action executed as the action share ν . Here we can identify three dominant states each with an action share of more than 20 %. Comparing these three, we see that they have similar send intervals while varying widely in decay factor, where a higher node density warrants a higher decay factor.

Table I: Optimal greedy policy π_g derived by dynRDF

N_{1H}	N_{2H}	Send interval i_0	Decay factor q	Action share ν
4	25	74 ms	1.80	21 %
4	29	85 ms	1.95	3 %
4	33	88 ms	2.15	4 %
8	25	67 ms	1.95	21 %
8	29	80 ms	1.70	9 %
8	33	74 ms	2.70	20 %
12	25	75 ms	2.10	4 %
12	29	68 ms	2.25	5 %
12	33	103 ms	2.10	12 %

VI. CONCLUSION

Position flooding is one promising solution to realize future UTM systems and unlock many UAV applications. While RDF is an effective protocol to facilitate this, our work proposes a mechanism to learn the optimal configurations for RDF based on the network size or the local UAVs density. Not only is our approach effective in obtaining configurations that outperform the default configuration obtained through brute-force by as much as 12 %, it also does so efficiently by only needing to explore 2.7 % of the full configuration space in as little as 20 policy iterations.

As safety is the highest priority in UAV operations, the real-world application of dynRDF would see UAVs equipped with a pre-determined policy obtained through simulation. Nevertheless, our proposed noise-based exploration mechanism also allows for ongoing learning in the real world, where the noise is tuned so that only policies that promise a sufficient performance (with an adequate safety buffer) are explored. By having continuous exploration active in the real-world deployment where tuples of state, actions and rewards are collected allows to detect drifts in the scenario and informs potential redeployment of a new policy. A setup like this represents an important step towards the deployment of RDF in the real world.

REFERENCES

- [1] M. Doole, J. Ellerbroek, and J. Hoekstra, “Estimation of traffic density from drone-based delivery in very low level urban airspace,” *Journal of Air Transport Management*, vol. 88, p. 101862, 2020.
- [2] K. Fuger, T. Marks, K. Kuladinitthi, and A. Timm-Giel, “Modeling of communication requirements for distributed utm using stochastic geometry,” in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, IEEE, 2024, pp. 666–671.
- [3] K. Fuger and A. Timm-Giel, “On the feasibility of position-flooding in urban uav networks,” in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, IEEE, 2023, pp. 1–5.
- [4] M. Randall, G. McMahon, and S. Sugden, “A simulated annealing approach to communication network design,” *Journal of Combinatorial Optimization*, vol. 6, no. 1, pp. 55–65, 2002.
- [5] A. A. P. Guimarães, I. M. Guerreiro, L. M. C. Sousa, D. C. Moreira, T. F. Maciel, and C. C. Cavalcante, “A (very) brief survey on optimization methods for wireless communication systems,” *International Telecommunications Symposium*, 2010.
- [6] I. Ahmad et al., “Machine learning meets communication networks: Current trends and future challenges,” *IEEE access*, vol. 8, pp. 223 418–223 460, 2020.
- [7] R. A. Nazib and S. Moh, “Reinforcement learning-based routing protocols for vehicular ad hoc networks: A comparative survey,” *IEEE Access*, vol. 9, pp. 27 552–27 587, 2021.
- [8] S. M. Akella, K. Mubasir, M. Michallik, S. R. Yeduri, and L. R. Cenkeramaddi, “Distributed q-learning-based uav-assisted small world wireless network for energy-efficient and delay-critical data transmission,” in *2023 2nd International Conference on 6G Networking (6GNet)*, IEEE, 2023, pp. 1–7.
- [9] A. Koushik, F. Hu, and S. Kumar, “Deep q-learning-based node positioning for throughput-optimal communications in dynamic uav swarm network,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 3, pp. 554–566, 2019.
- [10] M. M. Salah, R. S. Saad, R. M. Zaki, K. Rabie, and B. M. ElHalawany, “Multi-armed bandits for resource allocation in uav-assisted lora networks,” *IEEE Internet of Things Magazine*, vol. 8, no. 2, pp. 40–45, 2025.
- [11] R. Queiros, M. Kaneko, H. Fontes, and R. Campos, “Context-aware rate adaptation for predictive flying networks using contextual bandits,” *arXiv preprint arXiv:2504.05964*, 2025.
- [12] D. Pereira-Ruisánchez, Ó. Fresneda, D. Pérez-Adán, and L. Castedo, “Deep contextual bandit and reinforcement learning for irs-assisted mu-mimo systems,” *IEEE Transactions on Vehicular Technology*, vol. 72, no. 7, pp. 9099–9114, 2023.
- [13] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, “The broadcast storm problem in a mobile ad hoc network,” *Wireless networks*, vol. 8, no. 2, pp. 153–167, 2002.
- [14] H. Füßler, J. Widmer, M. Käsemann, M. Mauve, and H. Hartenstein, “Contention-based forwarding for mobile ad hoc networks,” *Ad Hoc Networks*, vol. 1, no. 4, pp. 351–369, 2003.
- [15] N. Peinecke, “How to stay well clear in corridors and swarms: Detect-and-avoid ranges for geovectoring concepts,” in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2022, pp. 57–63.
- [16] M. Collier and H. U. Llorens, “Deep contextual multi-armed bandits,” *arXiv preprint arXiv:1807.09809*, 2018.
- [17] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.
- [18] IEEE, “Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,” *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.
- [19] H. T. Friis, “A note on a simple transmission formula,” *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946.
- [20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.